48장 모듈

48.1 모듈의 일반적 의미

- **모듈**이란 애플리케이션을 구성하는 개별적 요소로서 재사용 가능한 코드 조각을 말한다.
  - 일반적으로 모듈은 기능을 기준으로 파일 단위로 분리
  - 이때 모듈이 성립하려면 모듈은 자신만의 파일 스코프를 가질 수 있어야 한다.
  - 자신만의 파일 스코프를 갖는 모듈의 자산(변수, 함수, 객체 등)은 기본적으로 비공개 상태다다시 말해, 자신만의 파일 스코프를 갖는 모듈의 모든 자산은 캡슐화되어 다른 모듈에서 접근할 수 없다.
  - 즉, 모듈은 개별적 존재로서 애플리케이션과 분리되어 존재한다.

export
모듈은 공개가 필요한 자산에 한정하여 명시적으로 선택적 공개가 가능

import
모듈 사용자는 모듈이 공개한 자산 중 일부 또는 전체를 선택해 자신의 스코프 내로 불러들여 재사용할수 있다.

- 공개된 모듈은 다른 모듈에서 재사용할 수 있다.
  - 이때 공개된 모듈의 자산을 사용하는 모듈을 모듈 사용자라 한다.

48.2 자바스크립트와 모듈

- 자바스크립트는 모듈 시스템을 지원하지 않는다.
  - 다시 말해, 자바스크립트는 모듈이 성립하기 위해 필요한 파일 스코프와 import, export를 지원하지 않았다.
- 자바스크립트는 파일마다 독립적인 파일 스코프를 갖지 않는다.

    - 다시 말해, 자바스크립트 파일을 여러 개의 파일로 분리하여 script 태그로 로드해도 분리된 자바스크립트 파일들은 결국 하나의 자바스크립트 파일 내에 있는 것처럼 동작한다.
- 즉, **모든 자바스크립트 파일은 하나의 전역을 공유**한다.
- 따라서 분리된 자바스크립트 파일들의 **전역 변수가 중복되는 등의 문제가 발생**할 수 있다.
    - 자바스크립트를 클라이언트 사이드, 즉 브라우저 환경에 국한하지 않고 범용적으로 사용하려는 움직임이 생기면서 모듈 시스템은 반드시 해결해야 하는 핵심 과제가 되었다.
- 이런 상황에서 제안된 것이 **CommonJS와 AMD**
- 이로써 자바스크립트의 모듈 시스템은 크게 CommonJS와 AMD로 나뉘게 되었고 브라우저 환경에서 모듈을 사용하기 위해서는 CommonJS 또는 AMD를 구현한 모듈 로더 라이브러리를 사용해야 하는 상황이 되었다.
- 자바스크립트의 런타임 환경인 Node.js는 모듈 시스템의 사실상 표준인 CommonJS를 채택했다.
    - Node.js 환경에서는 파일별로 독립적인 파일 스코프를 갖는다.

48.3 ES6 모듈(ESM)

- `ES6`에서는 클라이언트 사이드 자바스크립트에서도 동작하는 모듈 기능을 추가했다.
- `ES6` 모듈의 사용법은 간단하다.
    - script 태그에 `type="module"` 어트리뷰트를 추가하면 로드된 자바스크립트 파일은 모듈로서 동작한 다.
    - 일반적인 자바스크립트 파일이 아닌 ESM임을 명확히 하기 위해 ESM의 파일 확장하는 mjs를 사용할 것을 권장한다.

48.3.1 모듈 스코프

- ESM은 **독자적인 모듈 스코프**를 갖는다.
- ESM이 아닌 일반적인 자바스크립트 파일은 script 태그로 분리해서 로드해도

  독자적인 모듈 스코프를 갖지 않는다.
- 모듈 내에서 var 키워드로 선언한 변수는 더는 전역 변수가 아니며 window 객체의 프로퍼티도 아니다.

- 모듈 내에서 선언한 식별자는 모듈 외부에서 참조할 수 없다.
  - 모듈 스코프가 다르기 때문

48.3.2 export 키워드

- 모듈 내부에서 선언한 식별자를 외부에 공개하여

  다른 모듈들이 재사용할 수 있게 하려면 export 키워드를 사용한다.
- export 키워드는 선언문 앞에 사용한다.
  - 변수, 함수, 클래스, 등 모든 식별자를 export할 수 있게 된다.

```
// lib.mjs
// 변수의 공개
export const pi = Math.PI

// 함수의 공개
export function square(x){
    return x * x;
}

// 클래스의 공개
export class Person {
    constructor(name) {
        this.name = name;
    }
}
```

- exprot할 대상을 하나의 객체로 구성하여 한 번에 export할 수도 있다.

```
// lib.mjs
const pi = Math.PI

function square(x){
    return x * x;
}

export class Person {
    constructor(name) {
```

```
        this.name = name;
    }
}

// 변수, 함수, 클래스를 하나의 객체로 구성하여 공개
export { pi, square , Person};
```

48.3.3 import 키워드

- 다른 모듈에서 공개한 식별자를 자신의 모듈 스코프 내부로 로드하려면 import 키워드를 사용

```
// app.mjs
import { pi, square, Person } from './lib.mjs';
```

- 모듈이 export한 식별자 이름을 변경하여 import할 수도 있다.

```
import { pi as PI, square as sq, Person as P } from './lib.
mjs';
```

- 모듈에서 하나의 값만 export한다면 default 키워드를 사용할 수 있다.
- default 키워드를 사용하는 경우 기본적으로 이름 없이 하나의 값을 export 한다.

```
export default x => x * x;
```

- default 키워드와 함께 export한 모듈은 {} 없이 임의의 이름으로 import한다.

```
import square from './lib.mjs';
```